Package ‘GeneralizedWendland’

January 20, 2025

Type Package

Title Fully Parameterized Generalized Wendland Covariance Function

Version 0.6.0

Date 2024-02-18

Maintainer Thomas C. Fischer <thomascasparfischer@gmail.com>

Depends R (>= 3.1)

SystemRequirements gsl (>= 2.7)

Imports methods, stats, utils, Matrix, Rcpp, spam, spam64, parallel,
optimParallel, fields

LinkingTo Rcpp, RcppEigen, BH, Matrix

Suggests knitr, R.rsp, testthat (>= 3.0.0), mvtnorm, ggplot2,
gridExtra, dplyr, microbenchmark

Description A fully parameterized Generalized Wendland covariance function for use in Gaussian process models, as well as multiple methods for approximating it via covariance interpolation. The available methods are linear interpolation, polynomial interpolation, and cubic spline interpolation.
Moreno Bevilacqua and Reinhard Furrer and Tarik Faouzi and Emilio Porcu (2019) <url:<https://projecteuclid.org/journalArticle/Download?urlId=10.1214%2F1
AOS1652 >>.
Moreno Bevilacqua and Christian Caamaño-Carrillo and Emilio Porcu (2022) <arXiv:2008.02904>.
Reinhard Furrer and Roman Flury and Florian Gerber (2022) <url:<https://CRAN.R-project.org/package=spam >>.

RcppModules Wendland

License GPL (>= 2)

Encoding UTF-8

Config/testthat/edition 3

VignetteBuilder R.rsp, knitr

NeedsCompilation yes

Author Thomas C. Fischer [aut, cre],
Reinhard Furrer [aut, ths],
Josef Stocker [aut]

Repository CRAN

Date/Publication 2024-02-18 21:20:02 UTC

Contents

`GeneralizedWendland-package`

Fully Parametrized Generalized Covariance Function for R

Description

This package provides a fully parametrized generalized Wendland covariance function for use in geostatistical modeling, as well as various options for approximations and adjustments. In addition, the package comes with a modified process for parameter estimation, based on the spam-implementations, but compartmentalized using function factories.

Details

The package requires the user to complete some initial steps before it can be used. Apart from the R dependencies, the user will need to manually install the GNU Scientific Library on their system and ensure that R can find the required libraries. On Windows systems, users can install devtools and then install gsl via MSYS2 > pacman.

Author(s)

Thomas Caspar Fischer

See Also

`cov.wendland`.

`choleskyFactory`	*Function Factory for Generating chol Function with unified arguments*

Description

A function factory which returns a function of the form `function(Sigma)` which performs a cholesky decomposition using an approach tailored to the type of input Sigma. Currently works for vectors, matrices, spam objects, and dgCMatrix objects from the **Matrix** package.

Usage

```
choleskyFactory(chol.args = list(), Rstruct = NULL)
```

Arguments

`chol.args`	A list of optional settings for a cholesky function.
`Rstruct`	A `spam.chol.NgPeyton` object which represents the sparsity structure of covariance matrix Sigma.

Details

The output of `choleskyFactory` is intended to replace calls to `chol.default` or `chol.spam`. The object type is determined during runtime, after which the appropriate function is called to obtain the cholesky decomposition. For spam objects, the function attempts to use `update.spam.chol.NgPeyton` if `Rstruct` is specified, and upon failure defaults to `chol.spam`. The result is then assigned in the execution environment of `choleskyFactory`, so that `Rstruct` will be defined in the next call.

Value

A function of the form `function(Sigma)`.

Author(s)

Thomas Caspar Fischer

References

Hadley Wickham (2015) *Advanced R*, CRC Press. Reinhard Furrer and Roman Flury and Florian Gerber (2022) spam: SPArse Matrix, R package version 2.8-0.

See Also

`chol`, `chol.spam`, `update.spam.chol.NgPeyton`

Examples

```
set.seed(1234)
locations <- data.frame(x = runif(10), y = runif(10))
theta <- c(0.5,1,1,0,0)
dmat <- as.matrix(dist(locations))
Sigma <- cov.wendland(dmat, theta)

cholFun <- choleskyFactory(chol.args = list())
cholD <- cholFun(Sigma)

cholFun <- choleskyFactory(chol.args = list(pivot = TRUE))
cholD_pivot <- cholFun(Sigma)

cholFun <- choleskyFactory(chol.args = list(pivot = "RCM"))
cholS_RCM <- cholFun(spam::as.spam(Sigma))
```

`cov.askey` *Askey covariance function*

Description

A covariance function of the form

$$\rho_{\beta,\kappa,\mu} = \begin{cases} \sigma + \theta \quad 0 \leq r < \epsilon \\ (1-r)^{\mu} \quad \epsilon \leq r < 1 \\ 0 \quad 1 \leq r \end{cases}$$

where $r = h/\beta$. This is equivalent to the generalized Wendland covariance with $\kappa = 0$, but much more computationally efficient.

Usage

```
cov.askey(h, theta, ..., cov.args = list())
```

Arguments

`h`	A numeric vector, matrix, or spam object storing distances.
`theta`	Numeric vector $\vec{\theta} = (\beta, \sigma, \mu, \theta)$ storing parameters.
`...`	Other arguments.
`cov.args`	Named list of arguments. See Details.

Details

Using the list cov.args, users can provide the following arguments:

cov.eps (default: `.Machine$double.eps^0.5`**)** The threshold distance ϵ below which the function will return $\sigma + \theta$.

Value

Returns an object of the same type as input object h which stores the computed covariance values, i.e. a spam object if input h was also a spam object.

Author(s)

Thomas Caspar Fischer

References

Moreno Bevilacqua and Tarik Faouzi and Reinhard Furrer and Emilio Porcu (2019) Estimation and prediction using generalized Wendland covariance functions under fixed domain asymptotics, *Annals of Statistics*, **47**(2), 828–856.

See Also

`cov.wendland`

`cov.wendland` *Generalized Wendland Covariance Function*

Description

A fully parametrized generalized Wendland covariance function for use in geostatistical modeling, as well as multiple methods of obtaining computationally inexpensive approximations.

$$\rho_{\beta,\kappa,\mu} = \begin{cases} \sigma + \theta \quad 0 \leq r < \epsilon \\ \frac{\sigma}{B(1+2\kappa,\mu)} \int_r^1 (u^2 - r^2)^\kappa (1-u)^{\mu-1} du \quad \epsilon \leq r < 1 \\ 0 \quad 1 \leq r \end{cases}$$

where $r = h/\beta$

Usage

```
cov.wendland(h, theta, ..., cov.args = list())
```

Arguments

`h`	A numeric vector, matrix, or spam object storing distances.
`theta`	Numeric vector $\vec{\theta} = (\beta, \sigma, \kappa, \mu, \theta)$ storing parameters.
`...`	Other arguments.
`cov.args`	Named list of arguments. See Details.

Details

Using the list cov.args, users can provide the following additional arguments:

numint.abstol (default: `1e-3`**)** Absolute tolerance for numerical integration.

numint.reltol (default: `1e-3`**)** Relative tolerance for numerical integration.

numint.qag_key (default: `0`**)** Method to use in QAG integration (Values 1 - 6)

numint.subintervals (default: `0`**)** Number of subintervals to use in QAG/QAGS integration.

interp.method (default: `'none'`**)** Method to use for covariance interpolation. Valid methods are 'none', 'linear', 'polynomial', and 'cspline'.

interp.num_support (default: `0`**)** Number of support points to use for covariance interpolation.

cov.reparameterize (default: `TRUE`**)** Whether to apply the reparameterization $\mu = \frac{1+d}{2} + \kappa + \nu$, where ν takes the place of μ in input vector $\vec{\theta}$. This allows users to use box constraints in maximum likelihood estimation, as the covariance function is valid for $\nu \in [0, \infty)$ rather than $\mu \in [\frac{1+d}{2} + \kappa, \infty)$.

cov.eps (default: `.Machine$double.eps^0.5`**)** The threshold distance ϵ below which the function will return $\sigma + \theta$.

cov.d_value (default: `2`**)** Dimensionality of space in which measurements were taken. This only takes effect if `cov.reparameterize` is TRUE.

Value

Returns an object of the same type as input object h which stores the computed covariance values, i.e. a spam object if input h was also a spam object.

Author(s)

Thomas Caspar Fischer

References

Moreno Bevilacqua and Tarik Faouzi and Reinhard Furrer and Emilio Porcu (2019) Estimation and prediction using generalized Wendland covariance functions under fixed domain asymptotics, *Annals of Statistics*, **47**(2), 828–856.

Examples

```
h <- seq(0, 1, 0.01)
plot(0, type = "n", xlab = "Distance", ylab = "Covariance",
     xlim = c(0, 1), ylim = c(0,1))

theta <- c(range=1, sill=1, kappa=1, mu=0, nugget=0)
cov.args <- list()
lines(x = h, y = cov.wendland(h, theta, cov.args = cov.args),
      lwd = 2)

theta <- c(range=1, sill=1, kappa=1, mu=0, nugget=0)
cov.args <- list(cov.reparametrize = FALSE, cov.d_value = 2)
```

```
theta[4] <- (1 + cov.args[[2]])/2 + theta[3] + theta[4]
lines(x = h, y = cov.wendland(h, theta, cov.args = cov.args),
      col = "red", lty = 3, lwd = 3.5)

theta <- c(range=0.5, sill=1, kappa=1, mu=0, nugget=0)
cov.args <- list(interp.method="cspline", interp.num_support=100)
lines(x = h, y = cov.wendland(h, theta, cov.args = cov.args),
      col = "green", lwd = 2)

legend("topright", legend = c("Default", "No reparameterization",
                              "Cubic spline interpolation"),
       col = c(1, 2, 3), lty = c(1,3,1), lwd = c(2, 3.5, 2))
```

covarianceFactory *covarianceFactory*

Description

A function factory which sets up a covariance function.

Usage

```
covarianceFactory(covariance, cov.args = list())
```

Arguments

covariance	A function which takes as input an object containing distances (h), a vector of parameters (theta), and a list of optional settings (cov.args)
cov.args	A list of optional settings for a covariance function.

Details

`covarianceFactory()` is a function factory which takes as input an arbitrary covariance function and a list of additional arguments and returns a function with these arguments. The argument `cov.args` mainly serves to pass the relevant arguments to the covariance function, but also allows users to specify the following two arguments:

cov.fixed_range_value (default: `NA`**)** A constant value for the range parameter. Note that the code assumes that the first value of `theta` corresponds to the range parameter, as this notation is used in the **spam** package.

cov.fixed_nugget_value (default: `NA`**)** A constant value for the nugget parameter. Note that the code assumes that the last value of `theta` corresponds to the nugget parameter, as this notation is used in the **spam** package.

Note that if either of these arguments are specified, the corresponding entries should also be omitted in `theta`.

Value

A function of the form `function(h, theta)`. This function is enclosed in the execution environment of `choleskyFactory` and hence has access to the arguments `covariance` and `cov.args`. The manufactured function returns the result of `covariance(h = h, theta = theta, cov.args = cov.args)`.

Author(s)

Thomas Caspar Fischer

References

Wickham, H. (2015) *Advanced R*, CRC Press.

See Also

`covmat` and `cov.wendland`

Examples

```
library(GeneralizedWendland)
library(spam)

hs <- seq(0, 1, 0.05)
covSph <- covarianceFactory(cov.sph)
covExp <- covarianceFactory(cov.exp)
covWend <- covarianceFactory(cov.wendland)

plot(0, type = "n", xlim = c(0, 1), ylim = c(0, 1))
lines(hs, covSph(hs, theta = c(0.5, 1, 0)))
lines(hs, covExp(hs, theta = c(0.5, 1, 0)), col = "red")
lines(hs, covWend(hs, theta = c(0.5, 1, 0.5, 0.5, 0)), col = "green")
```

`factory-covDiag` *Diagnostics For Arbitrary Covariance Functions*

Description

A suite of diagnostic tools. The functions described here provide the user with quick access to diagnostics for arbitrary target covariance functions and arbitrary reference covariance functions.

Usage

```
covDiagFactory(target_covariance,
  diagnostic_funs = c("accumulated_error", "point_diagnostics"),
  reference_covariance = cov.askey, reference_cov.args = list())

accumulated_error(target_covFun, target_cov.theta, reference_covFun,
  reference_cov.theta, ..., absolute = TRUE, lower = 0, upper = 1,
  subdivisions = 500L, abs.tol = .Machine$double.eps^0.5,
  rel.tol = .Machine$double.eps^0.25)

point_diagnostics(target_covFun, target_cov.theta, reference_covFun,
  reference_cov.theta, ..., grid_resolution = 100)
```

Arguments

`diagnostic_funs`	A character vector for specifying which diagnostics to compute and return. Currently, `accumulated_error` and `point_diagnostics` are implemented. Note that the functions themselves are not intended to be called directly by the user.
`reference_covariance`	Covariance function with formals `function(h, theta, ..., cov.args)`
`reference_covFun`	Same as reference_covariance, but implicitly assumes that the input was generated using `covarianceFactory()`.
`reference_cov.args`	List with additional arguments to be passed to reference_covariance.
`reference_cov.theta`	Numeric vector containing parameters for reference_covariance.
`target_covariance`	Covariance function with formals `function(h, theta, ..., cov.args)`
`target_cov.theta`	Numeric vector containing parameters for target_covariance.
`target_covFun`	Same as target_covariance, but implicitly assumes that the input was generated using `covarianceFactory()`.
`...`	Other arguments
`grid_resolution`	Number of points n to evaluate the covariance function at.
`absolute`	Logical value. Whether to return absolute value.
`lower`	Lower boundary for accumulated error. Defaults to 0. Passed to `integrate()`.
`upper`	Upper boundary for accumulated error. Defaults to 1. Passed to `integrate()`.
`subdivisions`	The maximum number of subintervals. Passed to `integrate()`.
`abs.tol`	Absolute accuracy. Passed to `integrate()`
`rel.tol`	Relative accuracy. Passed to `integrate()`

Details

The function manufactured by `covDiagFactory` has the form `function(target_theta_list, target_args_list = list(), reference_cov.theta = NULL, ...)` and serves to iterate over a large variety of parameters.

target_theta_list List of named numeric vectors, each providing at least one or more values for a parameter.

target_args_list (default = `list()`**)** List of named vectors, each providing at least one or more values for each argument to be provided to target_covariance.

reference_cov.theta (default = `NULL`**)** A numeric vector with parameters for the reference covariance. This is primarily intended to use when comparing different covariance functions, in which case the target covariance is compared to the reference with constant parameters.If this argument is left at default and the target and reference are identical, the parameters in `target_theta_list` will be used by both. Otherwise this will raise an exception.

Value

`covDiagFactory()` is a function factory which generates a function for computing diagnostics across a large variety of parameter values and additional arguments. The manufactured function has the form `function(target_theta_list, target_args_list = list(), reference_cov.theta = NULL, ...` and returns a list of the same length as `diagnostic_funs`, each entry storing a `data.frame` with the results returned by the associated diagnostic function.

Available diagnostic functions

All diagnostic functions have in common that, apart from the required parameters kappa and mu, any remaining parameters are optional and captured using the `...` operator. Any unspecified arguments are left at default. The provided arguments are then turned into a grid using `expand.grid`, which is then iterated over to compute the requested diagnostic metrics. Note that there are some built-in checks which ensure that invalid configurations are dropped from the grid. For example, rows with interpolator = 'none' and a non-zero number of supports are dropped to prevent redundant computations.

Accumulated Error: The accumulated error corresponds to the area between the approximated and exact covariance curve.

Point diagnostics: This option returns error metrics across a user-defined number of points on the interval [0,1]. The metrics returned are "error", "absolute error", "maximum error", "target covariance value", and "reference covariance value".

Author(s)

Thomas Caspar Fischer

See Also

`cov.wendland`

Examples

```
interpolators <- c("linear", "cspline", "polynomial")
diagnostics <- c("accumulated_error", "point_diagnostics")

diagnosticFun <- covDiagFactory(cov.wendland, diagnostic_funs = diagnostics,
  reference_covariance = cov.wendland)
target_theta_list <- list(range = 0.5, sill = 1, kappa = c(0, 0.5, 1), mu = 0,
  nugget = 0)
```

```
target_args_list <- list(interp.method = interpolators, interp.num_support = 25)
wendland_comparison <- diagnosticFun(target_theta_list = target_theta_list,
  target_args_list = target_args_list)

diagnosticFun <- covDiagFactory(cov.wendland, diagnostic_funs = diagnostics,
  reference_covariance = cov.askey)
target_theta_list <- list(range = 0.5, sill = 1, kappa = 0, mu = 0.5, nugget = 0)
target_args_list <- list(interp.method = interpolators, interp.num_support = 25)
askey_comparison <- diagnosticFun(target_theta_list = target_theta_list,
  target_args_list = target_args_list, reference_cov.theta = c(0.5, 1, 0.5, 0))
```

factory-predict *Function factory for prediction generator*

Description

A function factory which generates a function for sampling from the predictive distribution

Usage

```
predictionFactory(y, locs0, locs1, covariance, X0 = list2DF(nrow = nrow(locs0)),
  X1 = list2DF(nrow = nrow(locs1)), ..., cov.args = list(), chol.args = list(),
  use_spam = TRUE)
```

Arguments

`y`	Observed measurements.
`locs0`	Locations at which measurements were obtained.
`locs1`	Locations at which to predict.
`covariance`	Covariance function.
`X0`	Covariates. By default corresponds to an empty data.frame() with `nrow(locs0)` rows.
`X1`	Covariates for new locations. By default corresponds to an empty data.frame() with `nrow(locs1)` rows.
`...`	Additional arguments. Unused.
`cov.args`	Additional arguments for covariance function.
`chol.args`	Additional arguments for function used to compute cholesky decomposition.
`use_spam`	Logical value. If TRUE, use spam to compute predictions.

Value

Returns a function of the form `function(n, param)`, where n corresponds to the number of simulations to run, and param is a parameter vector as returned by `optim`.

Author(s)

Thomas Caspar Fischer

Examples

```
set.seed(23)
n <- 100
res <- c(20, 20)

locs <- data.frame(x = runif(n), y = runif(n))
locs_new <- expand.grid(x = seq(0, 1, length.out = res[1]),
                        y = seq(0, 1, length.out = res[2]))
range <- 0.3
dmat <- as.matrix(dist(locs))
theta <- c(range, 1, 1, 0, 0)
cov.args <- list()
chol.args <- list()
Sigma <- cov.wendland(h = dmat, theta = theta, cov.args = cov.args)
y <- c(spam::rmvnorm(n = 1, Sigma = Sigma))

predictionFun <- predictionFactory(y = y, locs0 = locs, locs1 = locs_new,
  covariance = cov.wendland, cov.args = cov.args, chol.args = chol.args,
  use_spam = FALSE)

predictions <- predictionFun(n=10, param=theta)

image(x = seq(0, 1, length.out=res[1]),
      y = seq(0, 1, length.out=res[2]),
      z = matrix(apply(predictions,2,mean), res[1], res[2]),
      col = hcl.colors(9, "Blue-Red"),
      breaks = qnorm(seq(0.05, 0.95, 0.1)),
      xlab = "", ylab = "", xaxt = "n", yaxt = "n",
      useRaster = TRUE)
```

---

generalizedwendland-error

Simple errors defined by **GeneralizedWendland** *package*

Description

An overview of the `simpleError` objects defined by the **GeneralizedWendland** package.

Error definitions

`update.spam.chol.error`: ***"Updated covariance entries do not match length of original one."***
Warning which is raised when `update.spam.chol.NgPeyton` fails because the length of the entries in the updated covariance matrix is not equal to the length of the original matrix. This usually indicates that the initial sparsity structure was too sparse for the given parameters.

`wendland.insuffparam.error`: ***"Too few parameters for Wendland."*** Insert description here

`wendland.excessparam.error`: ***"Too many parameters for Wendland. Did you supply fix range or nugget?"*** Insert description here

`covfun.notfunction.error`: ***"Argument covariance must be a function."*** Insert description here

Author(s)

Thomas Caspar Fischer

`generalizedwendland-warning`

Simple warnings defined by **GeneralizedWendland** *package.*

Description

An overview of the `simpleError` objects defined by the **GeneralizedWendland** package.

Warning definitions

`update.spam.chol.warn`: ***"Updated covariance entries do not match length of original one. Deleting stored Rstruct."*** Warning which is raised when `update.spam.chol.NgPeyton` fails because the length of the entries in the updated covariance matrix is not equal to the length of the original matrix. This usually indicates that the initial sparsity structure was too sparse for the given parameters.

`wendland.interp.redundantsupport.warn`: ***"Argument interp.num_support > 0 while using exact method. Set to 0."*** Insert description here.

`wendland.interp.lowsupport.warn`: ***"Argument interp.method != 'none' with less than 3 support points. Forced to 'n***
Insert description here.

`wendland.interp.unimplemented.warn`: **"Interpolator not implemented. Forcing exact method."** Insert description here.

Author(s)

Thomas Caspar Fischer

`mleFactory` *Function Factory for Generating mle Function with unified arguments*

Description

A factory function which returns a function of the form `function(y, X = data.frame(), distmat, init_parameters, theta_llim, theta_ulim)` which can be called to compute the maximum likelihood estimates for a Kriging model.

Usage

```
mleFactory(covariance, cov.args = list(), chol.args = list(),
  optim.args = list(), hessian = FALSE, optimParallel.args = list())
```

Arguments

`covariance`	A function of the form `function(h, theta, ..., cov.args = list())`, where h is an object storing a distance matrix, theta is a numeric vector of parameters for the linear predictor and covariance function, and cov.args is a list of optional arguments for the covariance function.
`cov.args`	A list of optional settings for a covariance function.
`chol.args`	A list of optional settings for a cholesky function. *Note*: Valid input arguments change depending on whether the distance matrix provided to the output function is sparse. This may change in a future version.
`optim.args`	A list of optional settings for optim. See `optim` for documentation of valid arguments.
`hessian`	A logical value which specifies whether the hessian matrix is to be returned in the output. Is FALSE by default.
`optimParallel.args`	A list of optional settings for optimParallel. See `optimParallel` for documentation of valid arguments.

Details

The purpose of this function factory is to return an mle function with unified arguments. The returned function performs the same task as for example `spam::mle()`, but simplifies the process in two ways: The returned function detects whether the Gaussian process is a zero-mean process through the input argument X and whether methods from the **spam** package should be used based on the type of input argument distmat, and autonomously chooses appropriate methods to compute the neg2loglikelihood. Hence the user does not need to choose a specialized method themselves.

Value

A function of the form `function(y, X = data.frame(), distmat, beta0 = NULL, init_parameters, theta_llim, theta_ulim)` which returns the output of `optim` or `optimParallel` if `optimParallel.args` was specified.

Author(s)

Thomas Caspar Fischer

References

Hadley Wickham (2015) *Advanced R*, CRC Press.

See Also

`optim`, `optimParallel`, `covarianceFactory`, `choleskyFactory` and `optimFactory`

Examples

```
set.seed(57)
n <- 50
range <- 0.4
theta  <- c(range, 1, 1, 0, 0)

locs <- data.frame(x = runif(n), y = runif(n))
dmat  <- as.matrix(dist(locs))
Sigma <- cov.wendland(h = dmat, theta = theta)
y <- c(spam::rmvnorm(1, Sigma = Sigma))

init_parameters   <- c(0.7, 2, 0, 2, 2)
lower_constraints <- c(0.1, 0.1, 0, 0, 0)
upper_constraints <- c(sqrt(2), 2, 2, 2, 2)

mleFunction <- mleFactory(covariance = cov.wendland)
mle_result1 <- mleFunction(y = y, distmat = dmat,
                       init_parameters = init_parameters, theta_llim = lower_constraints,
                           theta_ulim = upper_constraints)

mleFunctionDM <- mleFactory(covariance = cov.wendland,
                            cov.args = list(fixed_range_value = range))
mle_result2 <- mleFunctionDM(y = y, X = data.frame(), distmat = dmat,
                             init_parameters = init_parameters[-1],
                             theta_llim = lower_constraints[-1],
                             theta_ulim = upper_constraints[-1])
```

`neg2loglikDiagFactory` *Diagnostics for arbitrarily specified, likelihood-based Gaussian process models*

Description

A helper function for rapidly exploring the parameter space around the maximum likelihood estimate

Usage

```
neg2loglikDiagFactory(y, X = data.frame(), distmat, covariance, ...)
```

Arguments

`y`	Dependent variable
`X`	Optional design matrix with covariates
`distmat`	Distance matrix. Can be provided either as a dense matrix or spam object.
`covariance`	Covariance function.
`...`	Other arguments to be passed on.

Details

theta_list Named list of vectors with parameters to be passed to covariance.

cov.args_list (default = `list()`**)** Named list of vectors with arguments to be passed to covariance

chol.args_list (default = `list()`**)** Named list of vectors with arguments to be passed to `choleskyFactory`.

Value

Returns a function of the form `function(theta_list, cov.args_list = list(), chol.args_list = list())` which returns a `data.frame` containing the neg2loglikelihood at all permutations of the provided arguments.

Note

The function manufactured by `neg2loglikDiagFactory` in principle also accepts covariance functions generated using `covarianceFactory`. However, the function is not yet compatible with the arguments `fixed_range_value` and `fixed_nugget_value`. For now, these should be left at default when using `covarianceFactory`.

Author(s)

Thomas Caspar Fischer

Examples

```
set.seed(63)
n <- 50
range <- 0.7
theta  <- c(range, 1, 1, 0, 0)

locs <- data.frame(x = runif(n), y = runif(n))
dmat  <- as.matrix(dist(locs))
Sigma <- cov.wendland(h = dmat, theta = theta)
y <- c(spam::rmvnorm(1, Sigma = Sigma))

neg2loglikIterator <- neg2loglikDiagFactory(y = y, distmat = dmat,
  covariance = cov.wendland)
```

```
theta_list <- list(range = 0.5, sill = 1, kappa = 0, mu = c(0, 0.25, 0.5),
  nugget = 0)
cov.args_list <- list(numint.abstol = c(1e-1, 1e-3, 1e-6), numint.reltol = c(1e-3))

results <- neg2loglikIterator(theta_list, cov.args_list = cov.args_list)
```

`neg2loglikFactory` *Function Factory for* `neg2loglikelihood` *with Unified Input Arguments*

Description

A function factory which generates a function of the form `function(parameters)` which returns the neg2loglikelihood.

Usage

```
neg2loglikFactory(y, X, distmat, covariance = NULL, cov.args = list(),
  chol.args = list(), Rstruct = NULL, covarianceFunction = NULL,
  choleskyFunction = NULL)
```

Arguments

`y`	Numeric vector. Dependent variable.
`X`	Optional data.frame containing covariates.
`distmat`	Distance matrix, either a numeric matrix or a spam object.
`covariance`	A function which takes as input an object containing distances (h), a vector of parameters (theta), and a list of optional settings (cov.args).
`cov.args`	A list of optional settings for a covariance function.
`chol.args`	A list of optional settings for a cholesky function.
`Rstruct`	A 'spam.chol.NgPeyton' object which represents the sparsity structure.
`covarianceFunction`	A function returned by `covarianceFactory()`.
`choleskyFunction`	A function returned by `choleskyFactory()`.

Details

This function factory returns a function of the form `function(parameters)` which computes the neg2loglikelihood for given input parameters. The purpose of this is to reduce the number of arguments that need to be specified by the user in a call to `optim`, or `optimParallel`. Furthermore, the function detects whether the input distmat is a spam object, and autonomously selects the appropriate method for computing the neg2loglikelihood.

The function is intended to be called from within `mleFactory`, but is also exported by NAMESPACE for users wishing to make use of the function. There are two distinct strategies available for using the function.

Option 1: the user may specify covariance, cov.args, chol.args, and Rstruct in the call. This syntax is more in line with the corresponding functions found in the **spam** package, yet still allows passing arguments for customizing the behaviour of the cholesky decomposition.

Option 2: the user may instead specify covarianceFunction and choleskyFunction, obtained from calls to `covarianceFactory` and `choleskyFactory`, respectively.

In both cases, the arguments y, X, and distmat are required input. Note that the two options are equivalent, apart from the second option allowing for more concise code.

Value

Returns function of the form `function(parameters)`.

Author(s)

Thomas Caspar Fischer

References

Hadley Wickham (2015) *Advanced R*, CRC Press.

See Also

`covarianceFactory` and `choleskyFactory`

Examples

```
set.seed(63)
n <- 50
range <- 0.7
theta  <- c(range, 1, 1, 0, 0)

locs <- data.frame(x = runif(n), y = runif(n))
dmat  <- as.matrix(dist(locs))
Sigma <- cov.wendland(h = dmat, theta = theta)
y <- c(spam::rmvnorm(1, Sigma = Sigma))
X <- data.frame()

neg2loglikFun <- neg2loglikFactory(y = y, X = X, distmat = dmat,
  covariance = cov.wendland, cov.args = list(), chol.args = list())
result1 <- neg2loglikFun(theta)

covarianceFun <- covarianceFactory(cov.wendland, cov.args = list())
choleskyFun <- choleskyFactory(chol.args = list())
neg2loglikFun <- neg2loglikFactory(y = y, X = X, distmat = dmat,
  covarianceFunction = covarianceFun, choleskyFunction = choleskyFun)
result2 <- neg2loglikFun(theta)
```

`optimFactory` *Function Factory for Optimization function with Unified Arguments*

Description

A function factory which returns a function with unified input arguments, and provides compatibility with the package **optimParallel**.

Usage

```
optimFactory(optim.args = list(), hessian = FALSE, optimParallel.args = list())
```

Arguments

`optim.args`	A named list of optional settings for optim. See `optim` for documentation of valid arguments.
`hessian`	A logical which specifies whether the hessian matrix is to be returned with the output.
`optimParallel.args`	A named list which is passed to `optimParallel` from the **optimParallel** package. See details.

Details

The function factory `optimFactory()` returns a function of the form `function(par, fn, gr = NULL, ..., lower, upper)`. It is intended to replace calls to `optim` or `optimParallel` by wrapping both functions. By default, it returns a function that corresponds to `optim` with default arguments.

To use optimParallel, users may specify the following arguments in optimParallel.args:

num_cores (default = `NULL`**)** The number of cores to use during numerical optimization. Is NULL by default, which corresponds to using `stats::optim`. When `num_cores` is a numeric value, the actual number of cores is set to `min(detectCores()-1, num_cores)` to avoid accidentally overloading the user's system.

forward (default = `FALSE`**)** A logical value which controls whether `optimParallel` should use central difference approximation of the gradient (`FALSE`) or forward difference approximation (`TRUE`).

loginfo (default = `FALSE`**)** A logical value which controls whether `optimParallel` should return additional information about the optimization process. See `optimParallel`.

Value

A function of the form `function(par, fn, gr = NULL, ..., lower, upper)` which returns the output obtained from calls to `optim` or `optimParallel`

Author(s)

Thomas Caspar Fischer

References

Hadley Wickham (2015) *Advanced R*, CRC Press. Florian Gerber and Reinhard Furrer (2019) optimParallel: An R package providing a parallel version of the L-BFGS-B optimization method, *The R Journal*, **11**(1), 352–358

See Also

`optim` and `optimParallel`

Examples

```
library(GeneralizedWendland)
library(optimParallel)

set.seed(43)
n <- 50
range <- 0.4
dist_max <- 2
theta  <- c(range, 1, 1, 0, 0)

locs <- data.frame(x = runif(n, 0, sqrt(dist_max)),
                   y = runif(n, 0, sqrt(dist_max)))
dmat  <- spam::nearest.dist(locs, locs, delta = dist_max)
Sigma <- cov.wendland(h = dmat, theta = theta)
y <- c(spam::rmvnorm(1, Sigma = Sigma))

init_parameters   <- c(0.7, 2, 0, 2, 2)
lower_constraints <- c(0.1, 0.1, 0, 0, 0)
upper_constraints <- c(sqrt(2), 2, 2, 2, 2)

mleFunction <- mleFactory(covariance = cov.wendland)
(mle_result <- mleFunction(y = y, distmat = dmat, init_parameters = init_parameters,
                       theta_llim = lower_constraints, theta_ulim = upper_constraints))

mleFunctionPar <- mleFactory(covariance = cov.wendland, optimParallel.args = list(num_cores = 2))
(mle_result_par <- mleFunctionPar(y = y, distmat = dmat, init_parameters = init_parameters,
                       theta_llim = lower_constraints, theta_ulim = upper_constraints))
```

Rcpp_Wendland-class *Class "Rcpp_Wendland"*

Description

Rcpp class which serves as an interface to the C++ implementation of the generalized Wendland covariance function.

Details

Parts of the covariance function require C and C++ dependencies. This Rcpp class serves as an interface to these dependencies.

Methods

Constructor:

Constructor `wend <- new("Rcpp_Wendland")`: Creates an instance of the Rcpp_Wendland class.

Get/Set methods:

setParameters `wend$setParameters(range, sill, kappa, mu, nugget)`: Set Parameters

setEpsTol `wend$setEpsTol(eps)`: set numeric precision.

Integration options:

setIntegratorQNG `wend$setIntegrator(abstol, reltol, intervals=0, qag_key=0)`

Interpolation options:

setInterpolator `wend$setInterpolatorLinear(num_points, interp_type=0)`

Computation:

compute `wend$compute(d)`: Compute for single value.

computeVector `wend$computeVector(d)`: Compute for numeric vector.

computeMatrix `wend$computeMatrix(d)`: Compute for numeric matrix.

computeMSparse `wend$computeMSparse(d)`: Compute for dgCMatrix from **Matrix**.

computeSpam `wend$computeSpam(index, values)`: Compute for spam object, turned into triplet form using `spam::triplet`.

Author(s)

Thomas Caspar Fischer

References

Dirk Eddelbuettel (2013) Seamless R and C++ Integration with Rcpp. *Springer*, New York

See Also

`cov.wendland`

Index